黑客派

Redis 在 Docker 中的数据持久化

作者：liumapp

原文链接：https://hacpai.com/article/1568193367769

来源网站：黑客派

项目 GitHub 地址：github/booklet
<script async src="https://pagead2.googlesyndication.com/pagead/js/adsbygoogle.js"></script>
<!-- 黑客派PC帖子内嵌-展示 -->
<ins class="adsbygoogle" style="display:block" data-ad-client="ca-pub-5357405790190342" data-ad-slot="8316640078" data-ad-format="auto" data-full-width-responsive="true"></ins>
<script>
    (adsbygoogle = window.adsbygoogle || []).push({});
</script>
Redis 提供了两种不同的持久化方法来将数据存储到硬盘里面。一种方法叫快照（snapshotting RDB），它可以将存在于某一时刻的所有数据都写入硬盘里面。
另一种方法叫只追加文件（append-only file，AOF），它会在执行写命令时，将被执行的写命复制到硬盘里面。
这篇文章梳理了 Redis 两种持久化方法的知识点，并通过 Docker + Docker-Compose 进行环的模拟，来进行数据的备份与恢复等操作。
至于测试数据，我通过一个 python 脚本批量录入三百万条 key-value 键值对（会消耗 719.42M 内存，来源于 redis-cli info 信息），没有 python 环境的同学，可以使用我在项目里准备的另一个 sell 脚本
python 脚本代码：
<pre><code class="language-python highlight-chroma"># -*- coding: UTF-8 -*-
file write.py
author liumapp
github https://github.com/liumapp
email liumapp.com@gmail.com
homepage http://www.liumapp.com
date 2019/9/9
#
import redis

r = redis.Redis(host="127.0.0.1", port=6379, db=0, password="admin123")

print("开始插入三百万条数据，每10万条数据提交一次批处理")

with r.pipeline(transaction=True) as p:

class="highlight-p">:

value = <span class "highlight-mi">0

while value <span cl ss="highlight-o">not found render function for node [type=NodeHTMLEntity, Tokens=<]not ound render function for node [type=NodeHTMLEntity, Tokens=<] 3000000 :

print ("开始插入" + <span class= highlight-nb">str (value) + /span> "条数据")</ pan>

p .<span cla s="highlight-n">sadd ("key" + tr (alue), "valu " + str<sp n class="highlight-p">(value))

value += 1

if (value % 10 000) ==</s an> 0 :

p .<span cla s="highlight-n">execute ()

</code></pre>

<h2 id="RDB">RDB</h2>
RDB 持久化是通过创建快照来获得数据副本，即简单粗暴的直接保存键值对数据内容
要启用 RDB（并关闭 AOF），我们需要修改 Redis 的配置文件(./redis_config/redis.conf):

<pre><code class="highlight-chroma">requirepass admin123

save 60 1000

stop-writes-on-bgsave-error no

rdbcompression no

dbfilename dump.rdb

appendonly no

appendfsync everysec

no-appendfsync-on-rewrite no

auto-aof-rewrite-percentage 100

auto-aof-rewrite-min-size 64mb

dir /data/
</code></pre>

上述配置会通过 docker-compose 的配置，映射到 Redis 容器中并启用，具体在下面的实操中绍
<script async src="https://pagead2.googlesyndication.com/pagead/js/adsbygoogle.js"></scpt>
<!-- 黑客派PC帖子内嵌-展示 -->
<ins class="adsbygoogle" style="display:block" data-ad-client="ca-pub-5357405790190342" data-ad-slot="8316640078" data-ad-format="auto" data-full-width-responsive="true"></in>
<script>
    (adsbygoogle = window.adsbygoogle || []).push({});
</script>
<h3 id="RDB配置说明">RDB 配置说明</h3>
上述配置中与 RDB 相关的配置如下

  save: 多久执行一次自动快照操作 比如设置为 <code>save 60 1000</code> 那么就表示在 60 秒之内，如果有 1000 次写入的话，Redis 就会自动触发 BGSAVE 命令 般来说，我们都会希望 Redis 可以有一个固定的周期来创建快照，那么可以这样设置 <cod>save 900 1</code> ，意思就是让 Redis 服务器每隔 900 秒，并且至少执行了一次写入操作后，触发 BGSAVE 指令
  stop-writes-on-bgsave-error: 在创建快照失败后是否仍然继续执行写命令
  rdbcompression: 是否对快照文件进行压缩

    yes: 开启，这种情况下，Redis 会采用 LZF 算法对 rdb 文件进行压缩
    no: 关闭

  dbfilename: 快照文件名
  dir: 快照文件存放目录

<h3 id="RDB触发条件">RDB 触发条件</h3>
RDB 的触发条件会比 AOF 麻烦，大致可以分为以下几种：

  通过 redis-cli 等客户端直接发送指令： <code>BGSAVE</code> BGSAVE 令，会让 Redis 调用 fork 创建一个子进程在后台运行，子进程将会负责创建快照到磁盘中 <p在演示案例中，启动 Redis 的 docker 容器后，在 redis-cli 中输入 <code>BGSAVE</code> 后，够在。/redis_data 目录下生成一个 temp-17.rdb 文件（或者其他以 rdb 结尾的）
  通过 redis-cli 等客户端直接发送指令：<code>SAVE</code> SAVE 指令**（意跟配置中的 save 没有半毛钱关系）**，会让 Redis 主进程直接开始创建快照，但在创建快照的过中，Redis 不会响应其他命令请求 在演示案例中，启动 Redis 的 docker 容器后，在 redis cli 中输入 <code>SAVE</code> 后，能够在。/redis_data 目录下生成一个 temp-17.rdb 文件（者其他以 rdb 结尾的）
  通过配置项 <code>save</code> 进行触发 具体请参照上文的参数说明 /li>
  通过 SHUTDOWN 命令关闭 Redis 服务器时，Redis 会自动触发一个 SAVE 指令 </l>
  通过标准 TERM 信号 kill 掉 Redis 服务时，Redis 也会自动触发一个 SAVE 指令 </l>
  通过 Redis 主从服务器的复制请求 主服务器收到从服务器的复制请求时，会触一次 BGSAVE 指令(当且仅当主服务器没有子进程在执行 BGSAVE)

<h3 id="RDB-Docker实操">RDB-Docker 实操</h3>

 通过 docker-compose 启动 Redis 容器 docker-compose.yml 配置如下
<pre><code class="language-yaml highlight-chroma">   </spa
>version:<span clas
="highlight-w"> "2"<span class="highlight-w"

    services<
pan class="highlight-p">:
      redis<s
an class="highlight-p">:
        image
span class="highlight-p">: <span class="highli
ht-s1">'redis:3.2.11'
        restart
span class="highlight-p">: always<span class="
ighlight-w">
        hostname</sp
n>: redis<span clas
="highlight-w">
        container_nam
: redis<span
class="highlight-w">
        ports<
pan class="highlight-p">:
          - '6379:6379'
/span>
        command</sp
n>: redis-server<sp
n class="highlight-w"> /usr/local/etc/redis/redis.conf
        volumes</spa
>:
          - ./redis_config/redis.conf<span class="hi
hlight-p">:/usr/local/etc/redis/redis.conf
          - ./redis_data/
/data/
</code></pre> 我将 Docker 容器中的 Redis 服务所产生的备份文件，映射在宿主的。/redis_data 目录下
 修改 Redis 配置文件，使 AOF 生效，并关闭 RDB 这里将上面的 redis.conf 内复制替换到。/redis_config/redis.conf 文件中即可
 启动 Redis 服务，并观察 redis_data 目录下是否有 dump.rdb 文件生成，有生成，则证备份成功
 数据恢复的话，我们不需要做其他操作，只要确保该 dump.rdb 存在，Redis 便会自动去取其中的数据

<h2 id="AOF">AOF</h2>
AOF 持久化会将被执行的写命令写到 AOF 文件的末尾，以此来记录数据发生的变化。因此，Red

s 只要从头到尾重新执行一次 AOF 文件包含的所有写命令，就可以恢复 AOF 文件所记录的数据集。<
p>
要启用 AOF（并关闭 RDB），我们需要修改 Redis 的配置文件(./redis_config/redis.conf)
<pre><code class="highlight-chroma">requirepass admin123

#save 60 1000

stop-writes-on-bgsave-error no

rdbcompression no

dbfilename dump.rdb

appendonly yes

appendfsync everysec

no-appendfsync-on-rewrite no

auto-aof-rewrite-percentage 100

auto-aof-rewrite-min-size 64mb

dir /data/

</code></pre>

上述配置会通过 docker-compose 的配置，映射到 Redis 容器中并启用，具体在下面的实操中
绍
<h3 id="AOF配置说明">AOF 配置说明</h3>
上述配置中与 AOF 相关的配置如下

 appendonly: 是否启用 AOF

 yes: 启用 AOF
 no: 关闭 AOF

 appendfsync: 启用 AOF 后的数据同步频率

 alaways: 每个 Redis 写命令都要同步写入硬盘。这样做会严重降低 Redis 的速度 （不建
）
 everysec: 每秒执行一次同步，显式地将多个写命令同步到硬盘 （推荐，对性能没有太大
响）
 no: 让操作系统来决定应该何时进行同步。（不建议） Redis 将不对 AOF 文
执行任何显式的同步操作，如果用户的硬盘处理写入操作的速度不够快的话，那么当缓冲区被等待写
硬盘的数据填满时，Redis 的写入操作将被阻塞，并导致 Redis 处理命令请求的速度变慢

 no-appendfsync-on-rewrite：在对 AOF 进行压缩（也被称为重写机制）的时候能否执
同步操作

 yes: 不允许
 no: 允许

 auto-aof-rewrite-percentage：多久执行一次 AOF 压缩，单位是百分比
 auto-aof-rewrite-min-size：需要压缩的文件达到多少时开始执行 auto-aof-re
rite-percentage 跟 auto-aof-rewrite-min-size 需要配套使用，比如当我们设置 auto-aof-rewrite-
ercentage 为 100，设置 auto-aof-rewrite-min-size 为 64mb 时 Redis 会在 AOF 产生
文件比 64M 大时，并且 AOF 文件的体积比上一次重写之后至少增大了一倍（100%）才执行 BGRE

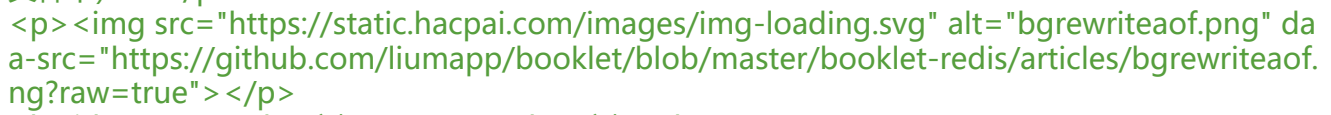
RITEAOF 重写命令 如果觉得 AOF 重写执行得过于频繁，我们可以把 auto-aof-rewrite-pecentage 设置 100 以上，比如 200，就可以降低重写频率 这里可以参考 Redis 的官方手，写的非常清楚：https://redislabs.com/ebook/part-2-core-concepts/chapter--keeping-data-safe-and-ensuring-performance/4-1-persistence-options/4-1-3-rewritingcomacting-append-only-files/
 dir：备份文件存放目录

<script async src="https://pagead2.googlesyndication.com/pagead/js/adsbygoogle.js"></script>
<!-- 黑客派PC帖子内嵌-展示 -->
<ins class="adsbygoogle" style="display:block" data-ad-client="ca-pub-5357405790190342" data-ad-slot="8316640078" data-ad-format="auto" data-full-width-responsive="true"></in>
<script>
    (adsbygoogle = window.adsbygoogle || []).push({});
</script>
<h3 id="AOF触发条件">AOF 触发条件</h3>
直接根据 appendfsync 的设置进行触发
<h3 id="AOF重写机制">AOF 重写机制</h3>
在上面的配置中，已经通过 auto-aof-rewrite-percentage 和 auto-aof-rewrite-min-size 两参数，简单介绍了 Redis 的 BGREWRITEAOF 重写命令
那么，为什么要用 AOF 重写机制呢？
因为 AOF 持久化是通过保存被执行的写命令来记录 Redis 数据库状态的，所以 AOF 文件随着系统运行会越来越大
而过于庞大的 AOF 文件会产生以下不良影响

 影响 Redis 服务性能；
 占用服务器磁盘空间；
 AOF 还原数据状态的时间增加；

所以 Redis 提供了一套 AOF 重写机制，通过创建一个新的 AOF 文件来替换掉旧的 AOF 文件，两个文件所保存的数据状态是相同的，但新的 AOF 文件不会包含冗余命令，所以体积会较旧 AOF 文小很多
但在实际的使用中，我们需要非常小心，不能让 Redis 的重写命令执行的过于频繁 (意：auto-aof-rewrite-percentage 的单位是百分比，值越大，重写频率越低，也千万别出现 0 这种) 因为 BGREWRITEAOF 的工作原理和 BGSAVE 创建快照的工作原理非常相似：Redis 会建一个子进程，然后由子进程负责对 AOF 文件进行重写，因为 AOF 文件重写也需要用到子进程，所快照持久化因为创建子进程而导致的性能问题和内存占用问题，在 AOF 持久化中也同样存在
更具体的 AOF 重写工作原理：

 Fork 主进程，产生一个带有数据副本的子进程在后台执行 Redis 这样设计可以保在重写过程中，不影响 Redis 主进程的服务正常运行，同时通过处理数据副本来保证数据的安全性*(注意，重写是针对数据副本来进行处理，而不是针对旧的 AOF 文件)**
 子进程 Fork 完成后，Redis 将启用 AOF 重写缓冲区，此刻开始，新的写入命令会被写入OF 缓冲区和 AOF 重写缓冲区中 这里启用的 AOF 重写缓冲区可以确保：在执行 AOF 重的过程中，任何新的写入命令产生，都不会导致新 AOF 文件的数据状态与 Redis 数据库状态不一致
 子进程完成对 AOF 文件的重写后，通知父进程
 父进程收到通知后，将 AOF 重写缓冲区的内容全部写入新的 AOF 文件中
 父进程将新的 AOF 文件替换掉旧的 AOF 文件**(注意，这一步会造成 Redis 阻塞，但问不大)**

BGREWRITEAOF 的工作流程图如下所示**(绘图源代码在项目的。/articles/bgrewriteaof.puml 文件下)**：


<h3 id="AOF-Docker实操">AOF-Docker 实操</h3>

 通过 docker-compose 启动 Redis 容器 docker-compose.yml 配置如下
<pre><code class="language-yaml highlight-chroma">
</code></pre> 我将 Docker 容器中的 Redis 服务所产生的备份文件，映射在宿主的。/redis_data 目录下
 修改 Redis 配置文件，使 AOF 生效，并关闭 RDB 这里将上面的 redis.conf 内复制替换到。/redis_config/redis.conf 文件中即可
 启动 Redis 服务，并观察 redis_data 目录下是否有 appendonly.aof 文件生成，有生成则证明备份成功 另外我们可以发现，3 百万条数据（700M）的备份文件，其实际占用磁空间约为 170M，这便是 Redis 重写机制强大的地方
 数据恢复的话，我们不需要做其他操作，只要确保该 appendonly.aof 存在，Redis 便会动去读取其中的数据

<h2 id="总结">总结</h2>
RDB 跟 AOF 都可以确保 Redis 的数据持久化，但各有特点
RDB 因为有默认的指令 SAVE 跟 BGSAVE 支持，所以比较适合对数据库做全量备份，比如每天晨 3 点开始执行一次 BGSAVE
而 AOF 因为是保存的写命令，因而更适合实时备份，事实上现在企业应用也基本都是采用的 AO
但光是使用了 RDB 或 AOF、甚至两个一起用，也还是不够的
对于一个需要支持可扩展的分布式平台而言，我们还需要提供一套复制备份机制，允许在一个周内，自动将 AOF 或者 RDB 的文件备份到不同的服务器下
这种情况下，我们就需要使用 Redis 的复制并生成数据副本功能，具体内容我会在下一篇文章进实操记录
<h2 id="参考链接">参考链接</h2>

https://redislabs.com/ebook/part-2-core-concepts/chapter-4keeping-data-safe-and-ensuring-performance
